SYSTEM AND METHOD FOR CREATING A BEST-MATCH OBJECT AT RUN TIME

BACKGROUND

[0001] Object-oriented technology continues to be an increasingly important tool for generating portable application code that can be readily used and reused. A basic premise of object-oriented technology is the use of objects. An object is a run-time entity with a specific set of instance methods and variables associated therewith.

[0002] In an effort to enhance the usability, portability, reliability and interoperability of objects, certain standards have been created. One group responsible for such standardization is referred to as the Object Management Group (OMG), which is a consortium of corporations, businesses and users interested in promoting object-oriented technology.

[0003] The Object Management Group has taken great steps in its standardization efforts. For example, the OMG is responsible for the creation of an object request broker (ORB), which is used to provide communications between clients and servers within a computing environment. The ORB is based upon an architecture touted by OMG and referred to as the Common Object Request Broker Architecture (CORBA).

[0004] One goal of the OMG is to provide distributed object-oriented applications and systems that coincide with the needs and desires of the ever-changing computing industry. This goal includes supporting communicatively coupled peripheral devices that may or may not have been supported at the time when an embedded or mobile-computing device was programmed.

Certificate of Mailing
I hereby certify that this Utility Patent Application is being deposited for delivery to Mail Stop: Patent Application, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, with the United States Postal Service **"EXPRESS MAIL POST OFFICE TO ADDRESSEE"** service under 37 CFR §1.10 on the date indicated below:

The envelope has been given U.S. Postal Service "Express Mail Post Office To Addressee" Package #EV269328944US.

Date: 11/5/2003

Signature

[0005] Although efforts have been made to meet the goals of the OMG, and of the object-oriented industry as a whole, further enhancements are still needed. For example, a need exists for an object-oriented computing environment that provides for the dynamic creation of objects to meet system needs.

SUMMARY

[0006] An embodiment of an object generator includes an object factory and a pool. The object factory polls proxies located in the pool to identify the proxy capable of producing a best-match object for a system need. The pool receives indicia of the desired object from the object factory and each of the proxies returns a confidence level responsive to the indicia. The proxy returning the greatest confidence level is directed to create the best-match object.

BRIEF DESCRIPTION OF THE DRAWINGS

[0007] Systems and methods for creating a best-match object at run time are illustrated by way of example and not limited by the implementations in the following drawings. The components in the drawings are not necessarily to scale emphasis instead is placed upon clearly illustrating the present systems and methods. Moreover, in the drawings, like reference numerals designate corresponding parts throughout the several views.

[0008] FIG. 1 is a functional block diagram of an embodiment of a computing device.

[0009] FIG. 2 is a functional block diagram of an embodiment of the object generator of FIG. 1.

[0010] FIG. 3 is a functional block diagram of an embodiment of the object factory of FIG. 2.

[0011] FIG. 4 is a flow chart illustrating an embodiment of a method for creating a best-match object at run time that can be implemented by the object generator of FIG. 2.

[0012] FIG. 5 is a flow chart illustrating an embodiment of a method for polling and selecting a best-match proxy that can be implemented by the object generator of FIG. 2.

[0013] FIG. 6 is a schematic diagram illustrating an embodiment of a method for creating a printer driver that can be implemented by the computing device of FIG. 1.

DETAILED DESCRIPTION

[0014] An object generator according to one embodiment of the invention includes an interface configured to receive information identifying a peripheral device that a user of a computing device would like to interface with in some way. The information may include a device identification string generated by the peripheral device and communicated to the object generator. In other embodiments, the information may include a device identification string or other identifier that only partially describes the peripheral device that the user would like to use. For example, when the peripheral device is a printer that is not configured to communicate a device identifier, a user of the computing device may be prompted to enter the name of the manufacturer, the model number, or some other information, such as the number and type of pens in the printer. The information provided by the user of the computing device is communicated to the object generator.

[0015] In response to receiving information identifying a peripheral device, the object generator polls a plurality of previously stored proxies that each represent the capability of a corresponding object that each individual proxy is designed to generate to the object generator. In one example, the separate proxies return a confidence level ranging from a minimum confidence level (i.e., the created object will very likely not be able to work with the peripheral device) to a maximum confidence level (i.e., the created object will be able to work with the peripheral device). A maximum confidence level might be communicated when the proxy is configured to produce an object designated for the identified peripheral device. When a maximum confidence level is identified, the object generator directs the proxy associated with the maximum confidence level to generate the object. The dynamically generated object can then be linked or otherwise made available to various applications operable on the computing device.

[0016] The object generator records a confidence level received from each of the proxies. In one embodiment, once a first confidence level is received, the object generator compares each subsequent confidence level with the confidence level received from the previous proxy. The object generator records an identifier associated with the proxy that returned the greater confidence level. Once all the

proxies have been polled, the object generator directs the proxy that provided the greatest confidence level to create an object for the computing device.

[0017] The object generator allows a software system to dynamically create different objects at run time based on a user input or information from other devices or software. The object generator and methods for creating a best-match object at run time enable the software system to generate and use objects that may not represent a confirmed match with a desired device. Stated another way, an object factory can generate and use a best-match object to handle a request, even when the system cannot determine with a complete certainty that the generated object is a match. This capability to generate and use a best-match object is especially useful for systems where a degraded result is preferred over no result at all and an error message.

[0018] The object generator is efficient in that information that describes what the object can do is encapsulated in a relatively small object proxy. The object generator can poll a plurality of object proxies without incurring the overhead associated with loading each object to determine if it is the correct object. In addition, the object generator is extremely flexible in that new object classes can be dynamically added to a software system by registering new object proxies with the object generator. New object proxies can be registered as they become available or in response to an object request. Moreover, the object generator is decoupled from the objects generated by the various object proxies.

[0019] Turning now to the drawings, reference is made to FIG. 1, which illustrates a functional block diagram of a computing device. Generally, in terms of hardware architecture, as shown in FIG. 1, computing device 100 includes a processor 110, memory 120, peripheral device(s) 130, network interface device(s) 140, and a user interface 150 that are communicatively coupled via local interface 160. The local interface 160 can be, for example but not limited to, one or more buses or other wired or wireless connections, as known in the art or that may be later developed. Local interface 160 may have additional elements, which are omitted for simplicity, such as controllers, buffers (caches), drivers, repeaters, and receivers, to enable communications. Further, local interface 160 may include address, control, and/or data connections to enable appropriate communications among the aforementioned components.

[0020] In the embodiment of FIG. 1, the processor 110 is a hardware device for executing software that can be stored in memory 120. The processor 110 can be any custom-made or commercially available processor, a central processing unit (CPU) or an auxiliary processor among several processors associated with the computing device 100. In some embodiments, processor 110 is a semiconductor-based microprocessor (i.e., a microchip) or an application-specific integrated circuit (ASIC).

[0021] Memory 120 includes any one or combination of volatile memory elements (e.g., random access memory (RAM, such as dynamic RAM or DRAM, static RAM or SRAM, etc.)) and nonvolatile memory elements (e.g., read-only memory (ROM), hard drives, tape drives, compact discs (CD-ROM.). Moreover, the memory 120 may incorporate electronic, magnetic, optical, and/or other types of storage media now known or later developed. Note that memory 120 can have a distributed architecture, where various components are situated remote from one another, but accessible by processor 110.

[0022] The software in memory 120 may include one or more separate programs, each of which comprises an ordered listing of executable instructions for implementing logical functions. In the example of FIG. 1, software in memory 120 includes an operating system 122, one or more application(s) 124, and an object generator 200. Application(s) 124 and object generator 200 function as a result of and in accordance with operating system 122. Operating system 122 controls the execution of the other application(s) 124 and computer programs, such as object generator 200, and provides scheduling, input-output control, file and data management, memory management, and communication control and related services.

[0023] Object generator 200 and application(s) 124 include one or more source programs, executable programs (object code), scripts, or other collections each comprising a set of instructions to be performed. As will be explained in detail below, object generator 200 includes logic that controls the execution of application(s) 124. More specifically, object generator 200 includes logic that controls the execution of an interface 210, object factory 220, and a pool 230 to dynamically create one or more objects that can be stored in object store 240 or linked to the application(s) 124. It should be well understood by one skilled in the art, after having become familiar with the object generator 200, that object generator 200 and application(s) 124 may be

written in a number of programming languages now known or later developed that support the creation and integration of objects. Moreover, object generator 200 and application(s) 124 may be stored across distributed memory elements in contrast with the single memory element (i.e., memory 120) shown in FIG. 1.

[0024] The peripheral device(s) 130 may take the form of human/machine devices such as but not limited to, a keyboard, a mouse or other suitable pointing device, a microphone, etc. Moreover, as illustrated in FIG. 1, the peripheral device(s) 130 can include laser printer 132, optical drive 134, and scanner 136, among other peripheral devices (not shown). Furthermore, the peripheral device(s) 130 may also include various models of known or later developed input/output devices, for example but not limited to, printers, display devices, etc.

[0025] Network-interface device(s) 140 include a host of devices that establish one or more communication sessions between computing device 100 and one or more local and/or wide area networks (not shown). Network-interface device(s) 140 may include but are not limited to, a modulator/demodulator or modem (for accessing another device, system, or network); a radio frequency (RF) or other transceiver; a telephonic interface; a bridge; an optical interface; a router; etc. For simplicity of illustration and explanation, these two-way communication devices are not shown.

[0026] It is often the case that computing device 100 is not entirely configured to operate the select device. For example, when the computing device 100 is a mobile-computing device that an operator desires to use in conjunction with a client's network. Generally, to print information stored or otherwise accessible via the mobile-computing device, the operator has to locate and identify a select printer available on the network. After identifying the select printer, many applications 124 require a printer specific device driver before the applications 124 can print information on the printer. As explained in further detail below, object generator 200 can be used to identify a best-match proxy configured to create an object, in the examples below, a driver that can be used to render data on a select printer.

[0027] User interface 150 enables an operator of the computing device 100 to selectively enter information that can be used by the object generator 200 to create a best-match object. User interface 150 forwards one or more parameters that identify a specific peripheral device 130 such as printer 132 that an operator of the computing

device 100 desires to use. Identifying parameters may include a manufacturer name, a model number, a trade name for a family of closely related devices, etc. When the peripheral device 130 is a printer that is not configured to forward an identification string to a coupled computing device, user interface 150 prompts the operator for information about the printer. For example, user interface 150 may prompt a user to enter a manufacturer, a series or family identifier, a model number, and may further ask the operator if the printer is capable of producing color or if the printer supports other functions. When an operator responds that the select printer is color capable, user interface 150 may prompt the operator for information regarding the printer pens installed in the printer.

[0028] When the peripheral device 130 is a printer such as a Hewlett-Packard DeskJet® 995c printer, which includes a Bluetooth® wireless interface, information identifying the printer and its capabilities can be communicated to computing device 100. The object generator 200 can then search for a best-match driver to support the DeskJet® 995c model printer. DeskJet® is the registered trademark of the Hewlett-Packard Company of Palo Alto, California, U.S.A. Bluetooth® is the registered trademark of Bluetooth Sig, Inc.

[0029] When the computing device 100 is in operation, the processor 110 is configured to execute software stored within the memory 120, to communicate data to and from the memory 120, and to generally control operations of the computing device 100 pursuant to the software. Operating system 122, one or more application(s) 124, and the object generator 200, in whole or in part, but typically the latter, are read by the processor 110, perhaps buffered within the processor 110, and then executed in accordance with the respective instructions.

[0030] As further illustrated in FIG. 1, object generator 200 includes an interface 210, an object factory 220, a pool 230, and an object store 240. Interface 210 receives an identification string generated by a peripheral device 130 or one or more information strings entered by an operator of computing device 100. Object factory 220 receives the identification string(s) and polls a plurality of object proxies stored in pool 230 to determine a best-match object proxy for communicating with or otherwise using the peripheral device 130. Each of the plurality of proxies stored in pool 230 is configured to return a confidence level within a confidence range to the object factory

220. The object proxy that responds with the highest confidence level that the object it will produce is the correct object for operating with the identified peripheral device 130 is directed by the object factory 220 to generate a corresponding object. The object generator 200 then stores the object in object store 240 or otherwise links the object to one or more applications 124.

[0031] It should be understood that object generator 200 can be embodied in any computer-readable medium for use by or in connection with an instruction execution system, apparatus, or device, such as a computer-based system, processor-containing system, or other system that can fetch the instructions from the instruction execution system, apparatus, or device, and execute the instructions. A "computer-readable medium" can be any methods and resources for storing, communicating, propagating, or transporting a program for use by or in connection with the instruction execution system, apparatus, or device. The computer-readable medium can be, for example but not limited to, an electronic, magnetic, optical, electromagnetic, infrared, or semiconductor system, apparatus, device, or propagation medium now known or later developed. Note that the computer-readable medium could even be paper or another suitable medium upon which the program is printed, as the program can be electronically captured, via for instance optical scanning of the paper or other medium, then compiled, interpreted, or otherwise processed in a suitable manner if necessary, and then stored in a computer memory.

[0032] Those skilled in the art will understand that various portions of object generator 200 can be implemented in hardware, software, firmware, or combinations thereof. In a preferred embodiment, object generator 200 is implemented using software that is stored in memory 120 and executed by a suitable instruction execution system. If implemented solely in hardware, as in an alternative embodiment, object generator 200 can be implemented with any or a combination of technologies well-known in the art (e.g., discrete logic circuits, application specific integrated circuits (ASICs), programmable gate arrays (PGAs), field programmable gate arrays (FPGAs), etc.), or technologies later developed.

[0033] In a preferred embodiment, the object generator 200 is implemented via a combination of software and data stored in memory 120 and executed and stored or otherwise processed under the control of processor 110. It should be noted, however,

that object generator 200 is not dependent upon the nature of the underlying processor 110 or memory 120 in order to accomplish designated functions.

[0034] FIG. 2 is a functional block diagram of an embodiment of the object generator of FIG. 1. As illustrated in the diagram, interface 210 of the object generator 200 receives input via connection 203 from a peripheral device 130 or via connection 205 from user interface 150. As indicated above, a device identifier received from a peripheral device 130 may include indicia of the manufacturer, model, and various capabilities of a select peripheral device 130. As also indicated above, when a peripheral device 130 is not configured to provide a device identifier, object generator 200 receives one or more identifiers as provided by a user of the computing device 100 via user interface 150. Once indicia of a select peripheral device has been provided, interface 210 forwards the indicia to object factory 220 via connection 215. Object factory 220 in turn polls the pool 230 via connection 225 to determine how many object proxies are available in pool 230 that are programmed to create objects that might be used by the peripheral device 130 that the operator would like to use. Pool 230 responds by indicating the number of available object proxies configured to generate objects associated with the select device type.

[0035] In some embodiments, pool 230 will contain related object proxies. Object proxies are related in that the underlying object that they represent can be associated with a peripheral device type. In other embodiments, pool 230 may hold one or more object proxies associated with multiple device types. For example, pool 230 includes object proxies 231, 232, 233, 234, . . . 237 configured to create objects associated with different printers. In other embodiments, pool 230 may include object proxies associated with scanners, cameras, external memory interface devices (e.g., an optical disk drive 134) among other peripheral device types.

[0036] Object factory 220 polls each of the identified object proxies, that is, proxy A 231, proxy B 232, proxy C 233, proxy D 234, through proxy *N* 237, where *N* is an integer representative of the number of related and available object proxies. Each object proxy responds by returning a confidence level corresponding with the likelihood that the underlying object that the object proxy represents will be able to meet the system need identified in the operator request. For example, when an operator of the computing device 100 desires to send information to printer 132, each of the separate proxies 231 -

237 generates and returns a confidence level to object factory 220 that corresponds to an expected probability that if the computing device 100 were to use the object generated from the corresponding proxy that the printer would produce an output commensurate with its capabilities. Object factory 220 then directs the object proxy that returned the highest confidence level via connection 225 to create an object. The select object proxy generates the object and forwards the object 245 along connection 235 to object store 240. The dynamically created object 245 can be copied, transferred, or otherwise linked via connection 248 to application(s) 124 operable on computing device 100. It should be understood that each of the various elements of object generator 200 illustrated and described in association with the functional block diagram of FIG. 2, including connections internal and external to object generator 200, can be implemented in both hardware, firmware, and software and any combination thereof. It should be further understood that in some embodiments object generator 200 can be distributed across one or more memory devices and or one or more instruction processing platforms.

[0037] FIG. 3 is a functional block diagram of an embodiment of the object factory 200 of FIG. 2. As shown in FIG. 3, object factory 220 includes a number of buffers or stores, comparators, and command generators. Specifically, object factory 220 includes buffer 310 coupled to receive indicia of a system need via input 215. As described above, the indicia can be in the form of one or more peripheral device identifiers generated by a peripheral device that an operator of the computing device 100 desires to use, or the indicia can be in the form of an alphanumeric string representing a device manufacturer, a model number, and in some cases, capabilities or functions of the peripheral device as entered by the operator into interface 150 (FIG. 1).

[0038] Buffer 310 is coupled to object factory engine 312 via connection 311. Object factory engine 312 receives the indicia and forwards the indicia via connection 313 to pool 230 (FIG. 2). In addition, object factory engine 312 generates and forwards a proxy pool request via connection 315 to pool 230. In response to the proxy pool request, the pool 230 returns an indication of the number of available proxies associated with the requested peripheral device type via connection 322. As illustrated in FIG. 3, the number of available proxies, N, is forwarded to proxy pool quantity store 330, which then forwards the indication via connection 331 to object factory engine 312.

[0039] As further illustrated in FIG. 3, object factory engine 312 generates a confidence level request, which is forwarded in turn to each of the *N* available proxies within pool 230 via connection 317. Each individual proxy, when polled via the confidence level request, generates and returns a confidence level via connection 348 ranging from a minimum confidence level, which relates to no confidence that an object created by the proxy will enable an application to use the identified peripheral device 130 (FIG. 1), to a maximum confidence level, which relates to a known match of the object to be created with the identified peripheral device 130. The received confidence level can be temporarily retained in confidence level store 360 before being forwarded to confidence level comparator 370 via input 365. Confidence level comparator 370 is configured to receive a maximum confidence level from maximum confidence level store 350 via input 352. In a first comparison operation, confidence level comparator 370 determines if the confidence level received from confidence level store 360 via input 365 matches the maximum confidence level received via input 352.

[0040] When it is the case that the confidence level received from confidence level store 360 matches the maximum confidence level, the confidence level comparator 370 is configured to store an identifier corresponding to the proxy that generated the maximum confidence level in best-match proxy store 380. Confidence level comparator 370 communicates the proxy identifier via connection 375 to the best-match proxy store.

[0041] When it is the case that the confidence level received from confidence level store 360 fails to match the maximum confidence level provided by maximum confidence level store 350, confidence level comparator 370 compares the confidence level received from the present proxy with a confidence level from a previously received proxy received via connection 373 from confidence level store 334. Confidence level comparator 370 determines which of the two inputs is associated with the highest confidence level and stores an associated proxy identifier in best-match proxy store 380 as well as the highest confidence level in confidence level store 334. Consequently, object factory 220 records both the highest reported confidence level and the proxy that reported the confidence level as the object factory 220 receives each proxy vote from the plurality of object proxies in pool 230.

[0042] As each proxy is polled by the object factory 220, counter comparator 345 receives a counter value via input 343 from counter 340 and the number of available

proxies, *N*, via input 335 from quantity store 330. Counter comparator 345 then determines if the value represented in counter 340 is equal to the number of available proxies. The value in counter 340 is incremented by 1 after each confidence level is received from a polled proxy. If the value is not equal, not all proxies have been polled by the object factory 220. Once counter comparator 345 determines that the value in counter 340 has reached the number of available proxies, command generator 390 receives an indication of the best-match proxy via input 385 and generates a command directing the best-match proxy to generate its corresponding object. Command generator 390 forwards the command to pool 230 (FIG. 2) via output 395.

[0043] Reference is now directed to FIG. 4, which includes a flow chart illustrating an embodiment of a method for creating a best-match object at run time that can be implemented by the object generator 200 of FIG. 2. As illustrated in FIG. 4, the method begins with object generator 200 receiving a request for an object as indicated in block 402. In response to the request, as illustrated in block 404, the object generator 200 polls a plurality of object proxies for a confidence level that each object proxy can generate the requested object. After polling each of the object proxies for a confidence level, object generator 200 selects one of the object proxies in an attempt to satisfy the request, as shown in block 406. Once the object generator 200 has selected an object proxy, the object generator 200 directs the select one of the plurality of object proxies to create the proxy as illustrated in block 408.

[0044] FIG. 5 is a flow chart illustrating an embodiment of a method for polling and selecting a best-match proxy that can be implemented by the object generator 200 of FIG. 2. As shown in FIG. 5, the method begins with block 502 where the object generator 200 loads an integer representing the number of available object proxies and initializes a counter. As indicated in block 504, the object generator 200 receives indicia of a desired peripheral device that the object to be created is intended to support.

[0045] Next, in block 506, the object generator forwards the indicia of the desired peripheral device. Thereafter, the object generator 200 directs a proxy to generate a confidence level responsive to the indicia as indicated in block 508. In block 510, the object generator 200 receives the confidence level from the proxy. As indicated in query 512, the object generator 200 then determines if the received confidence level is equivalent to a maximum confidence level. When the received confidence level is not

equal to a maximum confidence level, as indicated by the flow control arrow exiting query 512 labeled "NO," the object generator records the confidence level as indicated in block 514.

[0046] Processing continues with query 516 where the object generator 200 determines if the received confidence level exceeds a previously recorded confidence level. When the received confidence level exceeds a previously recorded confidence level, as indicated by the flow control arrow labeled "YES" exiting query 516, object generator records an identifier associated with the proxy that forwarded the received confidence level as shown in block 518. Otherwise, when it is the case that the received confidence level does not exceed a previously received confidence level as indicated by the flow control arrow labeled "NO" exiting query 516, the function in block 518, i.e., recording the proxy, is bypassed. Processing continues with object generator 200 performing query 520 to determine if all available proxies have provided an associated confidence level. When it is the case that not all proxies have been polled, as indicated by the flow control arrow labeled "NO" exiting query 520, the counter is incremented as indicated in block 522 and object generator 200 repeaters the functions of blocks 508 through 522 until one of the proxies indicates that it is a match, that is, when a proxy responds with a confidence level that is equal to the maximum confidence level as indicated by the flow control arrow labeled "YES" exiting query 512 or it is the case that all proxies have responded as indicated by the flow control arrow labeled "YES" exiting query 520.

[0047] Once either of query 512 or query 520 has been satisfied, processing continues with block 524 where object generator 200 directs the proxy that responded with the highest confidence level to create the object. Thereafter, the object is loaded or otherwise made available to the computing device 100 as indicated in block 526.

[0048] Any process descriptions or functions in the flow charts presented in FIGs. 4 and 5 should be understood to represent modules, segments, or portions of code or logic, which include one or more executable instructions for implementing specific logical functions in the associated process. Alternate implementations are included within the scope of the disclosed methods in which functions may be executed out of order from that shown or discussed, including substantially concurrently or in reverse order, depending on the functionality involved, as would be understood by those reasonably skilled in the art after having become familiar with the object generator

200 and methods for creating best-match objects at run time. For example, the embodiment of the method for creating a best-match object at run time illustrated in FIG. 5 indicates that object generator 200 initializes a counter and determines the number of available proxies in function 502 and receives indicia of a desired object in function 504. These functions can be performed in reverse order or substantially simultaneously as would be understood by those skilled in the art.

[0049] FIG. 6 is a schematic diagram illustrating an embodiment of a method for creating a printer driver that can be implemented by the computing device 100 of FIG. 1. As illustrated in the diagram, printer factory 610 searches for the best match among a plurality of printer driver proxies by communicating with the printer proxy interface 620 via connection 615. As indicated in the diagram, printer factory 610 uses a first data type 612 labeled, "<<FAMILY_HANDLE>>"to coordinate and monitor communications with printer proxy 620. Printer proxies, represented by interface 620 use a second data type labeled "<<MODEL_HANDLE>>"to coordinate and communicate details about supported device models.

[0050] As indicated in the diagram, printer factory 610 is configured to separately poll a first printer driver proxy 630 configured to generate a driver for a Hewlett-Packard DeskJet® 9XX-VIP printer via printer proxy interface 620. Upon receipt of a printer family name, a printer model name, or some other information that can be used to classify a set of printers, the printer factory 610 forwards the information to the first printer driver 630, via interface 620, along with an instruction for the printer driver proxy 630 to provide a confidence level via a proxy vote 627. When the identifying information indicates that the desired printer is a Hewlett-Packard DeskJet® 9XX-VIP series printer, the first printer driver proxy 630 will return a confidence level of 100 via the proxy vote 627. When the identifying information indicates that the desired printer is a Hewlett-Packard DeskJet® 9XX printer the first printer driver proxy 630 will return a confidence level below the maximum confidence level of 100. For example, first printer driver proxy 630 may return a confidence level of 70 or 90 depending on how close the model number (i.e., the characters "XX" in the model number of the identifier) matches the proxy.

[0051] Each of the remaining printer driver proxies 628, 626, 624, and 622 are similarly polled until either all are polled or one of the printer device proxies returns a maximum

confidence level to the printer factory 610. Once either of these two conditions has been satisfied, the printer factory 610 uses interface 620 to direct the printer device proxy that returned the highest confidence level to create an associated object, in this example, a printer driver. As indicated in the diagram, when printer device proxy 622 is directed to create an object, the printer device proxy 622 generates the printer driver labeled "ETC." 644 via connection 643. When printer device proxy 624 is directed to create an object, printer device proxy 624 generates the printer driver labeled "DJ6XX" 646 via connection 645. When printer device proxy 626 is directed to create an object, printer device proxy 626 generates the printer driver labeled "DJ8XX" 648 via connection 647. When printer device proxy 628 is directed to create an object, printer device proxy 628 generates the printer driver labeled "DJ9XX" 650 via connection 649. When printer device proxy 630 is directed to create an object, printer device proxy 630 generates the printer driver labeled "DJ9XXVIP" 652 via connection 651. Once the select object has been created by one of the available printer device proxies, a printer interface 640 representing the dynamically generated object is returned to the printer factory 610 using interface 620.

[0052] Blocks or functions in the block diagrams and schematics illustrated in FIGs. 1-3 and FIG. 6 should be understood to represent modules, segments, portions of code or logic, or data processed by logic. That is, the blocks or functions include one or more executable instructions for implementing specific logical functions in the associated process. Alternate implementations are included within the scope of the disclosed object generator 200 and methods for creating a best-match object in which functions may be executed out of order from that shown or discussed, including substantially concurrently or in reverse order, depending on the functionality involved, as would be understood by those reasonably skilled in the art.